Université de Bordeaux **Licence S&TS Mention Informatique, S3**
4TINA01U **2021/2022**

Programmation fonctionnelle
TP noté du Vendredi 17 Décembre
Durée : 1h20.

Le barème est donné à titre **indicatif**. Le sujet comporte 4 pages.

---

a) Télécharger le fichier `student.ml` depuis Moodle. Le travail doit être effectué dans ce fichier `student.ml` lequel doit être déposé sur Moodle à l'issue du TP. Ce fichier contient un barème indicatif.

b) Renseignez vos nom, prénom et numéro de groupe.

c) Pensez à sauver régulièrement votre travail.

d) Il est recommandé d'utiliser les fonctions du module `List` chaque fois que c'est approprié.

e) Les types des fonctions demandées sont explicités dans les exemples. Il y a des exemples pour chaque fonction demandée.

---

1. Écrire une version **récursive terminale** de la fonction `make_list n e` qui retourne une liste contenant `n` fois l'élément `e`.

```
utop[15]> make_list;;
- : int -> 'a -> 'a list = <fun>
utop[16]> make_list 0  3;;
- : int list = []
utop[17]> make_list 5 0;;
- : int list = [0; 0; 0; 0; 0]
utop[18]> make_list 3 true;;
- : int list = [true; true; true]
```

2. Écrire une fonction `make_zeros n` qui retourne une liste contenant `n` fois l'entier `0`.

```
utop[21]> make_zeros;;
- : int -> int list = <fun>
utop[22]> make_zeros 4;;
- : int list = [0; 0; 0; 0]
```

On suppose que les éléments d'une liste sont numérotés à partir de `0`.

3. En supposant que la liste `l` contient au moins `i + 1` éléments[1], écrire la fonction `get_nth l i` qui retourne l'élément numéro `i` de la liste `l`.

```
utop[31]> let l = [2; 4; 6; 8];;
val l : int list = [2; 4; 6; 8]
utop[32]> get_nth l 0;;
- : int = 2
utop[33]> get_nth l 3;;
- : int = 8
utop[34]> get_nth l 4;;
Exception: Failure "hd".
utop[35]> get_nth [] 0;;
Exception: Failure "hd".
```

[1] Dans le cas où la liste ne contient pas assez d'éléments, la fonction pourra produire une erreur.

4. En supposant que la liste `l` contient au moins `i + 1` éléments, écrire la fonction `set_nth l i e` qui retourne une liste identique à `l` sauf l'élément numéro `i` de la liste qui est remplacé par `e`.

```
utop[41]> set_nth;;
- : 'a list -> int -> 'a -> 'a list = <fun>
utop[42]> l;;
- : int list = [2; 4; 6; 8]
utop[43]> set_nth l 2 10;;
- : int list = [2; 4; 10; 8]
utop[44]> set_nth [] 2 10;;
Exception: Failure "tl".
utop[45]> set_nth l 4 10;;
Exception: Failure "tl".
```

5. Écrire la fonction `get_nth_opt l i` qui retourne `None` si `l` n'a pas de `i` eme élément et `Some e` où `e` est le `i` eme élément de `l`.

```
utop[54]> get_nth_opt;;
- : 'a list -> int -> 'a option = <fun>
utop[55]> get_nth_opt l 2;;
- : int option = Some 6
utop[56]> get_nth_opt l 5;;
- : int option = None
utop[57]> get_nth_opt [] 0;;
- : 'a option = None
```

Un jeu se joue sur une grille carrée représentée par une liste de listes d'entiers contenant initialement que des zéros.

```
          C
    | 0 1 2 3 4
  -------------
  0 | 0 0 0 0 0
  1 | 0 0 0 0 0
L 2 | 0 0 0 0 0
  3 | 0 0 0 0 0
  4 | 0 0 0 0 0
```

Une grille sera représentée par la **liste de ses lignes**, chacune des lignes étant elle-même représentée par une liste d'entiers. On utilisera donc le type suivant :

```
type grid = int list list
```

Par la suite, il n'est pas demandé de forcer le type `int list list` à être `grid`. Donc dans les exemples, votre code pourra afficher `int list list` au lieu de `grid`

6. Écrire la fonction `make_empty_grid size` **récursive terminale** qui fabrique une grille de taille `size`.[2]

[2] On pourra utiliser la fonction `make_zeros` vue précédemment.

```
utop[61]> make_empty_grid;;
- : int -> grid = <fun>
utop[62]> make_empty_grid 3;;
- : grid = [[0; 0; 0]; [0; 0; 0]; [0; 0; 0]]
utop[63]> make_empty_grid 4;;
- : grid = [[0; 0; 0; 0]; [0; 0; 0; 0]; [0; 0; 0; 0]; [0; 0; 0; 0]]
```

7. Écrire la fonction `grid_size grid` qui retourne la longueur du côté de la grille carrée `grid`.

```
utop[71]> grid_size;;
- : 'a list -> int = <fun>
utop[72]> let grid = make_empty_grid 3;;
val grid : grid = [[0; 0; 0]; [0; 0; 0]; [0; 0; 0]]
utop[73]> grid_size grid;;
- : int = 3
```

8. Écrire un prédicat `correct_z_p grid z` qui retourne vrai si l'entier `z` est un numéro de ligne ou de colonne correct pour la grille `grid`.

9. Écrire un prédicat `correct_coor_p grid li co` qui retourne vrai si `li` et `co` sont des numéros de ligne et de colonne corrects pour la liste `l`.[3]

```
utop[91]> grid;;
- : grid = [[0; 0; 0]; [0; 0; 0]; [0; 0; 0]]
utop[92]> correct_coor_p;;
- : 'a list -> int -> int -> bool = <fun>
utop[93]> correct_coor_p grid 2 1;;
- : bool = true
utop[94]> correct_coor_p grid 2 3;;
- : bool = false
```

10. Écrire la fonction `get_square grid line column` qui retourne l'entier qui se trouve dans la case `column` de la ligne `line`.

11. Écrire la fonction `set_square grid line column n` qui retourne la grille dans laquelle la case `column` de la ligne `line` a pris la valeur `n`.

```
utop[111]> get_square;;
- : 'a list list -> int -> int -> 'a = <fun>
utop[112]> set_square;;
- : 'a list list -> int -> int -> 'a -> 'a list list = <fun>
utop[113]> let grid = make_empty_grid 3;;
val grid : grid = [[0; 0; 0]; [0; 0; 0]; [0; 0; 0]]
utop[114]> let grid = set_square grid 1 2 5;;
val grid : grid = [[0; 0; 0]; [0; 0; 5]; [0; 0; 0]]
utop[115]> let grid = set_square grid 2 1 8;;
val grid : grid = [[0; 0; 0]; [0; 0; 5]; [0; 8; 0]]
utop[116]> get_square grid 1 2;;
- : int = 5
utop[117]> get_square grid 2 0;;
- : int = 0
```

[3] On pourra utiliser la fonction `correct_z_p` vue précédemment.

Voici la grille obtenue présentée ligne par ligne.

```
        C
    [[0; 0; 0];
L    [0; 0; 5];
     [0; 8; 0]]
```

12. Écrire le prédicat `pred_square_p predicat grid line column` qui retourne `true` si l'entier contenu dans la case `column` de la ligne `line` vérifie le prédicat `predicat`.

```
utop[101]> pred_square_p;;
- : (int -> 'a) -> grid -> int -> int -> 'a = <fun>
utop[121]> grid;;
- :grid = [[0; 0; 0]; [0; 0; 5]; [0; 8; 0]]
utop[122]> pred_square_p (fun x -> x mod 2 = 0) grid 1 2;;
- : bool = false
utop[123]> pred_square_p (fun x -> x > 4) grid 1 2;;
- : bool = true
```

13. Écrire un prédicat `zero_square_p grid line column` qui retourne `true` si l'entier contenu dans la case `column` de la ligne `line` est égal à `0` et `false` sinon.

```
utop[131]> grid;;
- : grid = [[0; 0; 0]; [0; 0; 5]; [0; 8; 0]]
utop[132]> zero_square_p grid 1 0;;
- : bool = true
utop[133]> zero_square_p grid 1 2;;
- : bool = false
```

14. Écrire une fonction `grid_map f grid` qui retourne une nouvelle grille de même taille dont les éléments sont ceux de la grille initiale auxquels on a appliqué la fonction `f`.[4]

```
utop[141]> grid;;
- : grid = [[0; 4; 0]; [0; 2; 5]; [0; 8; 3]]
utop[142]> grid_map pred grid;;
- : grid = [[-1; 3; -1]; [-1; 1; 4]; [-1; 7; 2]]
utop[143]> grid_map (fun x -> x * x) grid;;
- : grid = [[0; 16; 0]; [0; 4; 25]; [0; 64; 9]]
```

[4] On pourra utiliser la fonction `map` du module `List`.